Operations Count and Data Locality in AD

A. Lyons (Vanderbilt U.) / J. Utke (ANL)

"Minimizing operations counts and maximizing data locality
for efficient derivative codes in automatic differentiation"

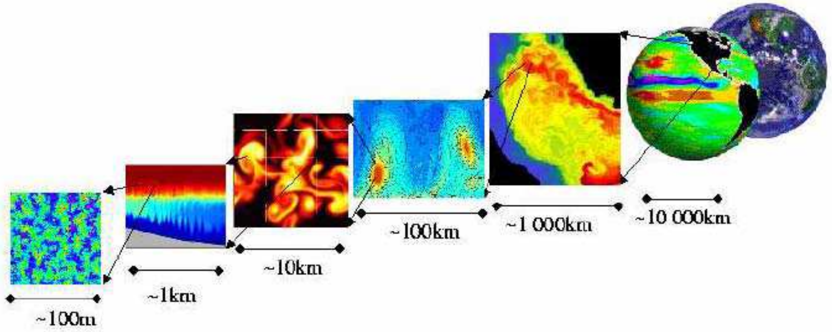
1. automatic differentiation (AD) and graphs
2. graph operations and code generation in AD
3. high level concerns (adjoints with checkpointing)
4. low level code generation has significant runtime effects
5. assumption 1: optimizing beasic block preaccumulations is significant
6. assumption 2: data locality is significant
7. assumption 3: code can be generated to help compiler optimization
8. $\Rightarrow$ heuristics
9. experiments and conclusions

bad

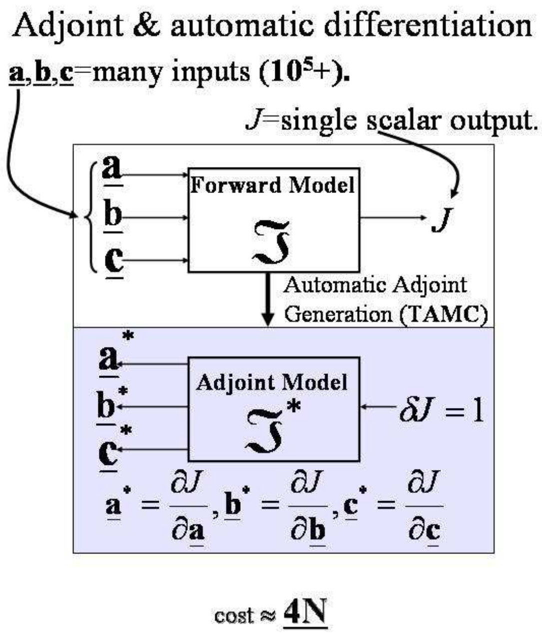
AD in general

- derivatives for numerical models (science, engineering)
- optimization, parameter estimation, sensitivity/uncertainty analysis
- need derivative information (gradients, Jacobian/Hessian vector products)
- large scale computation
- complexity/quality issues with finite differences

MIT General Circulation Model (ocean,atmosphere) ©Heimbach/Hill @ MIT



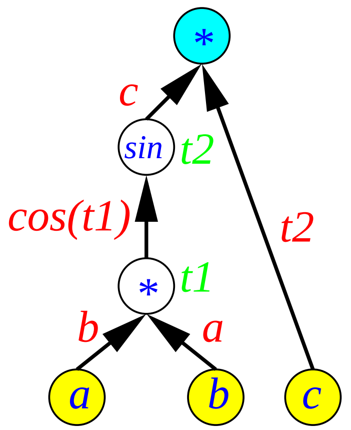
model scalable from single PC to 1000+ processor clusters



AD and graphs: a simple example

$f : y = sin(a * b) * c$

yields a graph representing the order of computation:

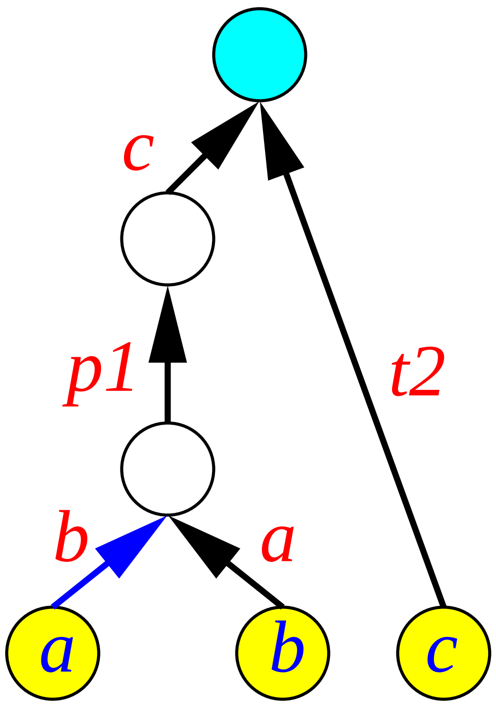


- use some temporaries $t1, t2$
- all intrinsics $\phi(\ldots, w, \ldots)$ have local partial derivatives $\frac{\partial\phi}{\partial w}$ as edge labels:
- may have to compute partials

```
t1 = a*b
p1 = cos(t1)
t2 = sin(t1)
y  = t2*c
```

academic

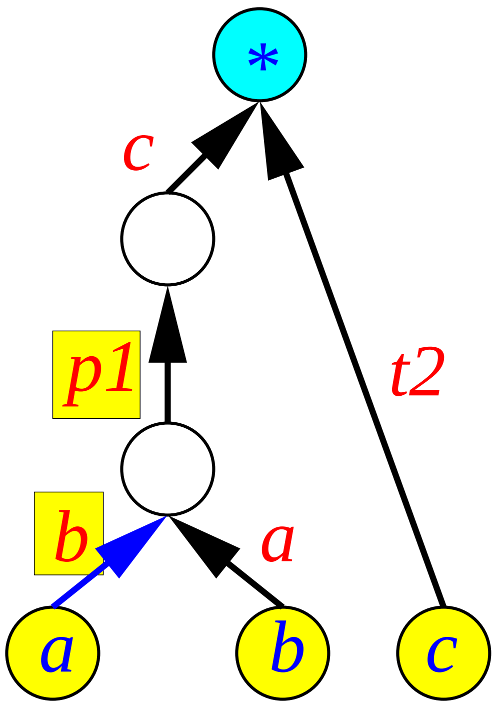
(local) Jacobians 1



edge elimination: pick an edge

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
```

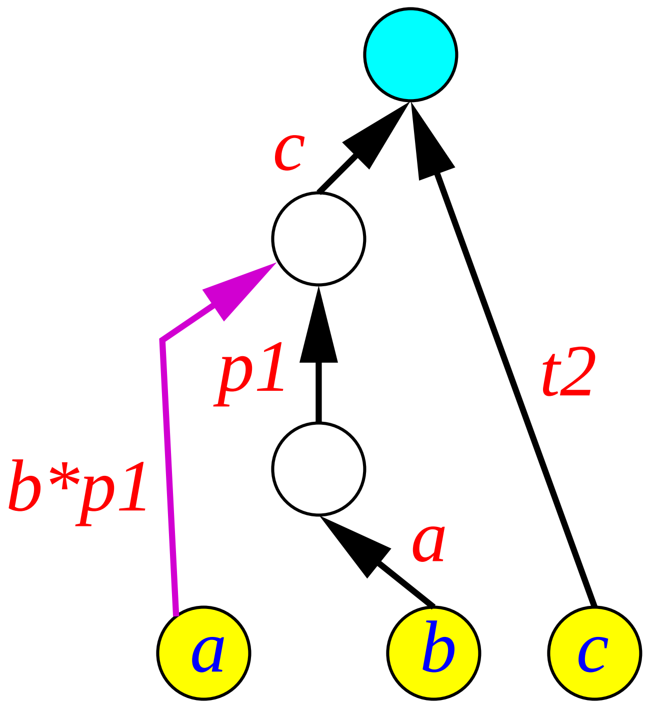
(local) Jacobians 2



edge elimination: front elimination: pairs with outgoing edges of target vertex

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
```

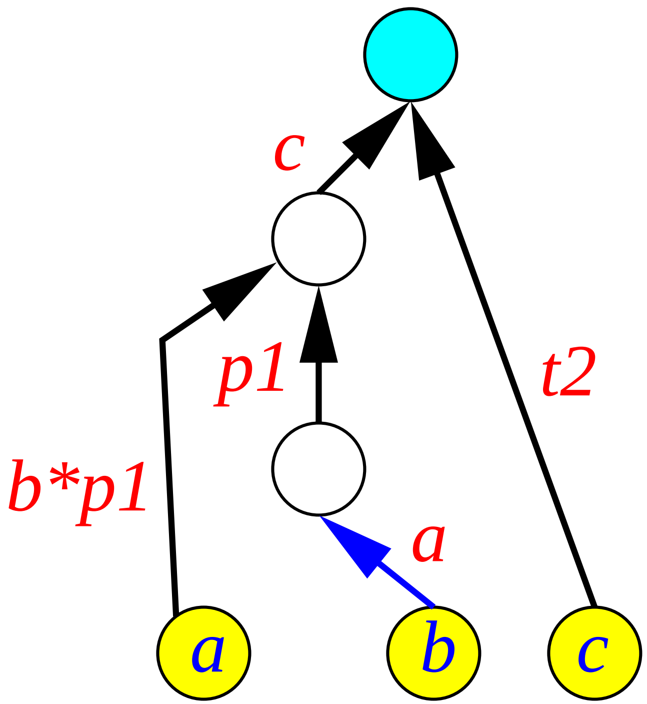
(local) Jacobians 3



edge eliminations: multiply edge labels and attach to edge with same source and target of the paired edge

```
t1    = a*b
p1    = cos(t1)
t2    = sin(t1)
y     = t2*c
z1    = b * p1
```

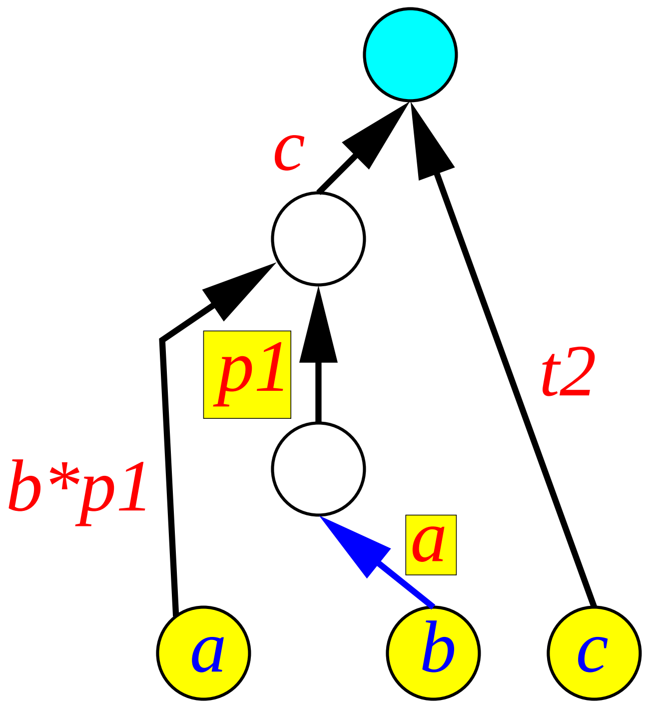
(local) Jacobians 4



edge eliminations: pick the next target

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
```

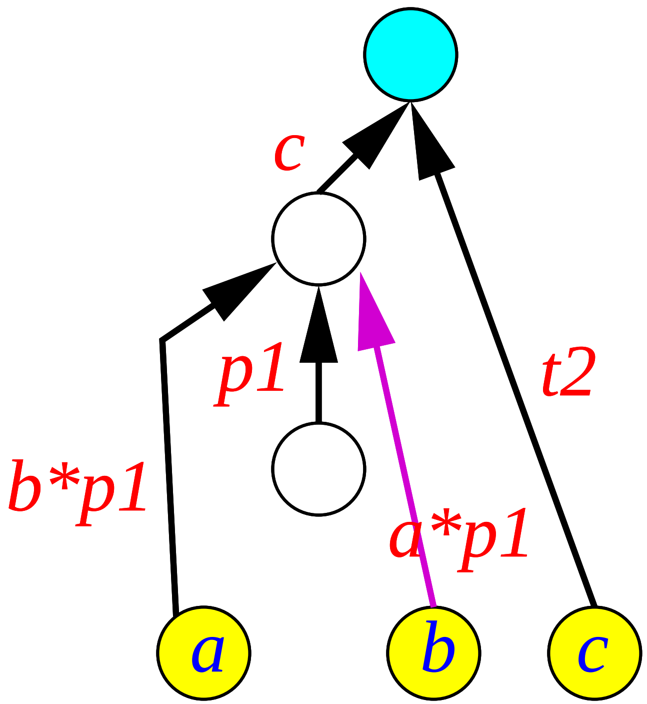
(local) Jacobians 5



edge eliminations: pair it up with the outgoing edges of the target vertex

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
```

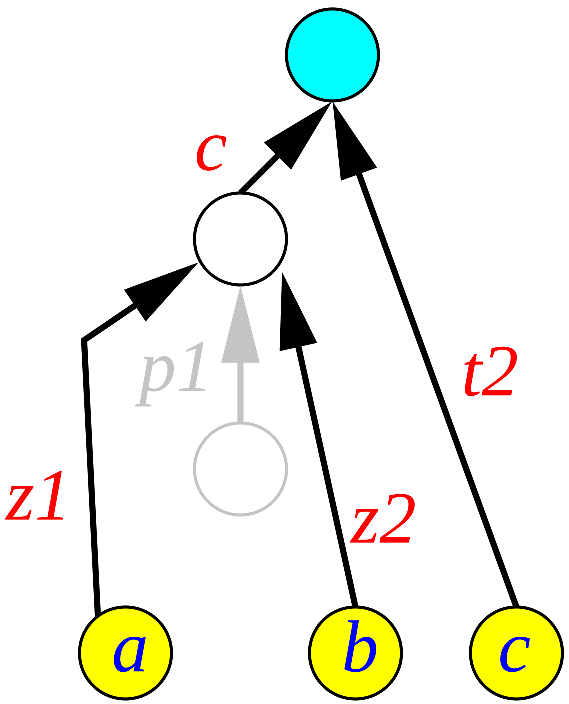
(local) Jacobians 6



edge eliminations: multiply the labels and attach the result

```
t1    = a*b
p1    = cos(t1)
t2    = sin(t1)
y     = t2*c
z1    = b * p1
z2    = a * p1
```

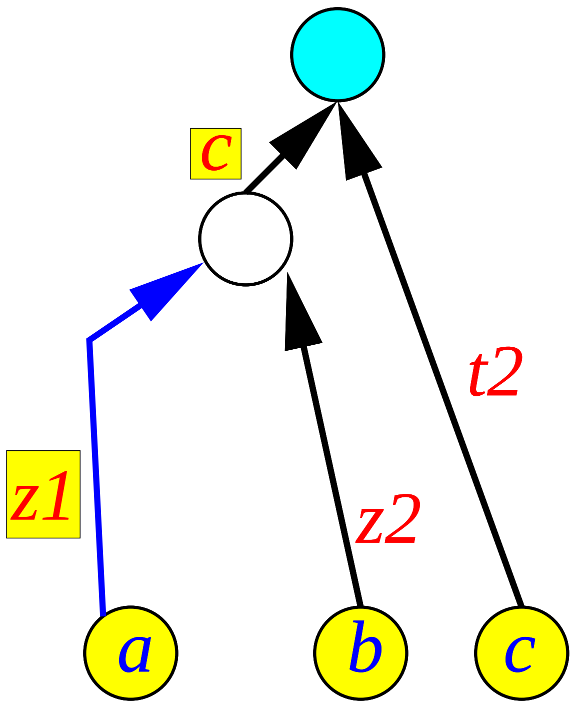
(local) Jacobians 7



edge eliminations: there is an isolatex vertex/edge that can be removed; rename edge labels

```
t1    = a*b
p1    = cos(t1)
t2    = sin(t1)
y     = t2*c
z1    = b * p1
z2    = a * p1
```

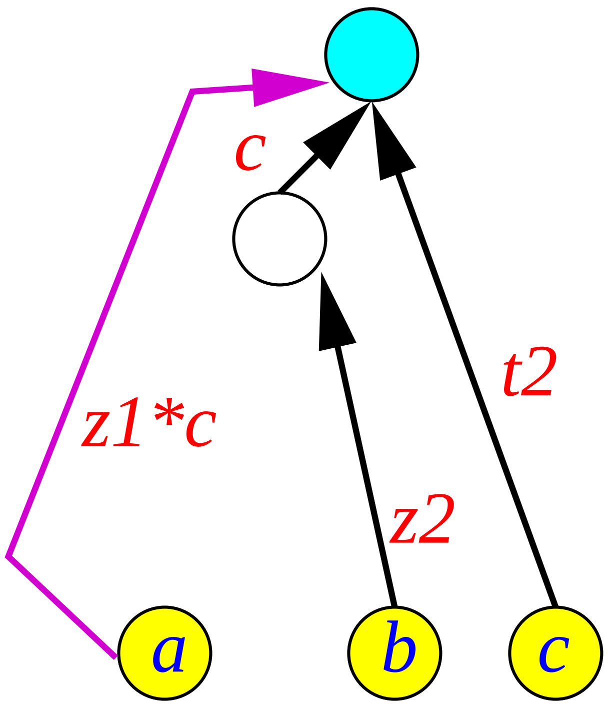
(local) Jacobians 8



edge eliminations: pick the next edge

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
z2   = a * p1
```

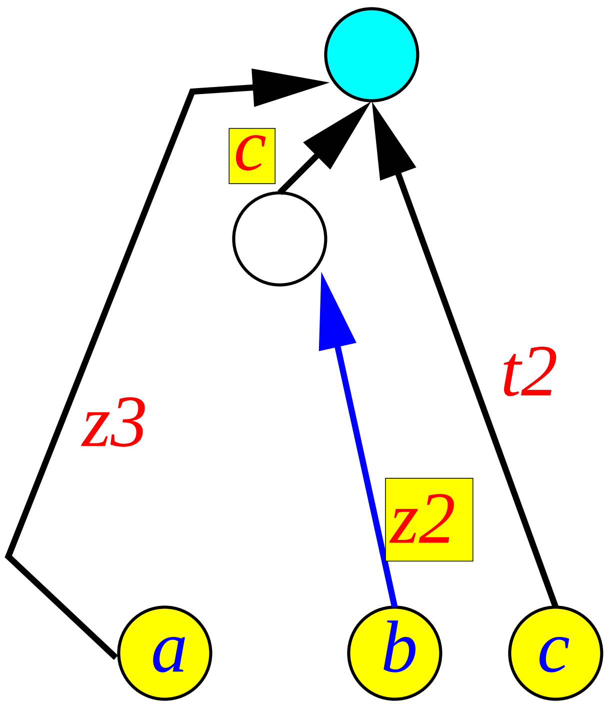
(local) Jacobians 9



edge eliminations: multiply labels etc.

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
z2   = a * p1
z3   = z1 * c
```

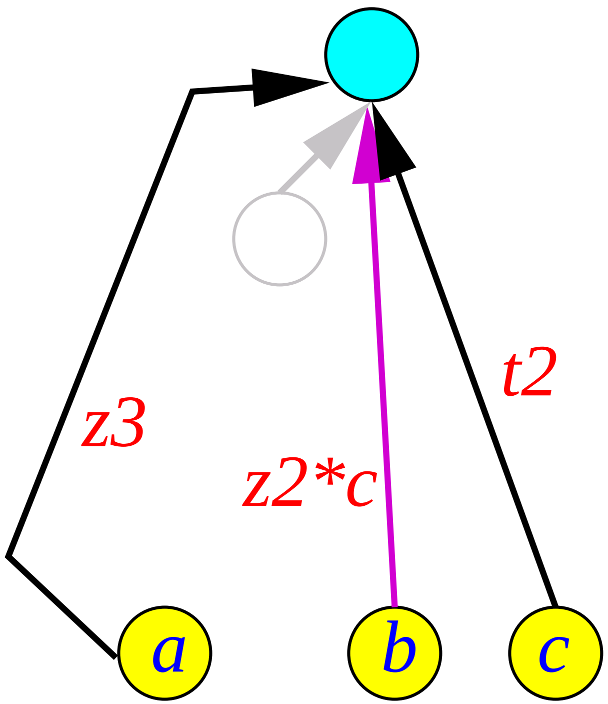
(local) Jacobians 10



edge eliminations: pick the next one

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
z2   = a * p1
z3   = z1 * c
```

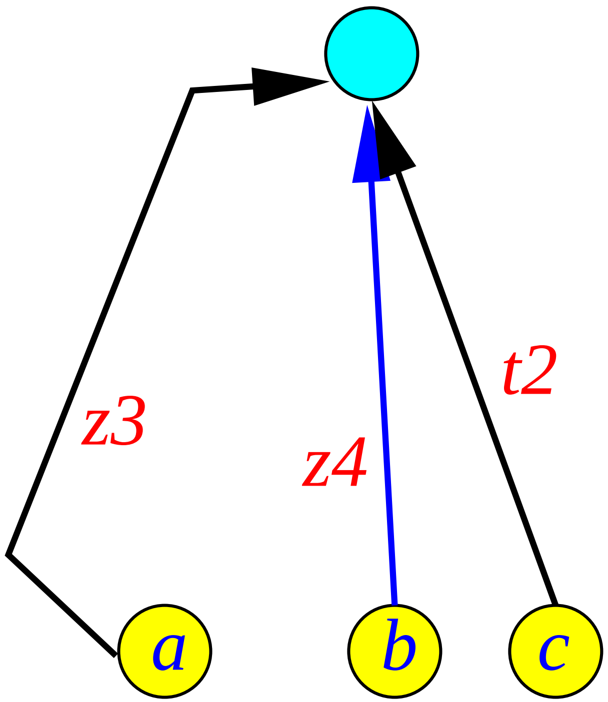
(local) Jacobians 11



edge eliminations: mutliply labels etc.

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
z2   = a * p1
z3   = z1 * c
z4   = z2 * c
```

(local) Jacobians 12

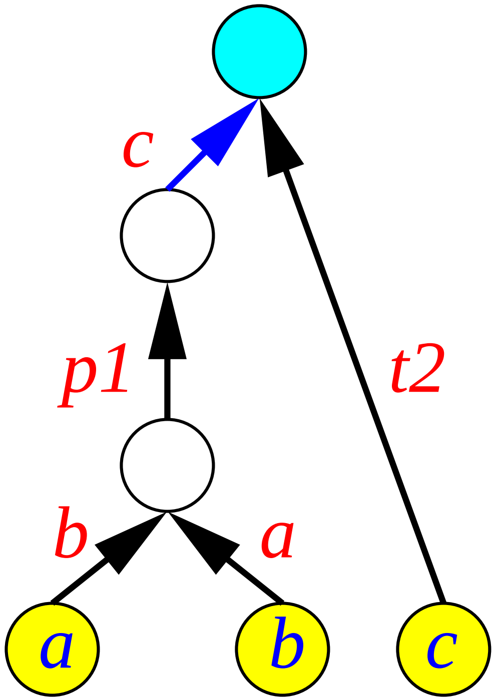


edge eliminations: bipartite graph,
done in 4 operations

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = b * p1
z2   = a * p1
z3   = z1 * c
z4   = z2 * c
```

jading

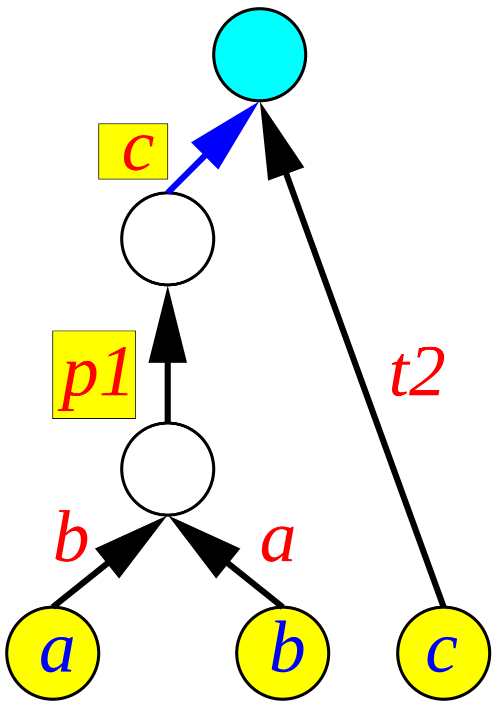
(local) Jacobians 13



edge elimination: pick an edge

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
```

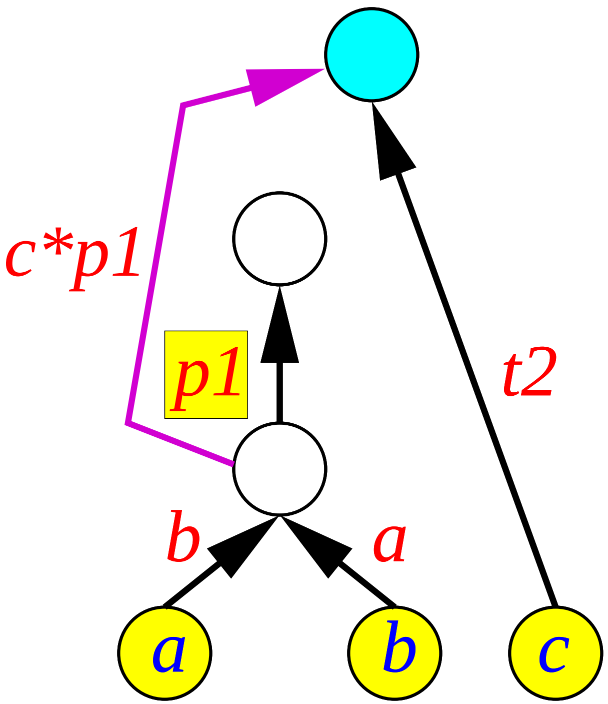
(local) Jacobians 14



edge elimination: back elimination: pairs with incoming edges of source vertex

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
```

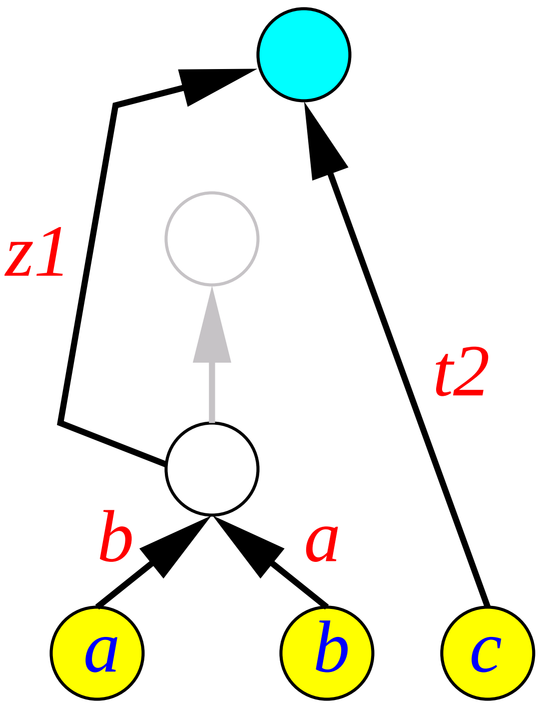
(local) Jacobians 15



edge eliminations: multiply edge labels and attach to edge with same target and source of the paired edge

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = c * p1
```

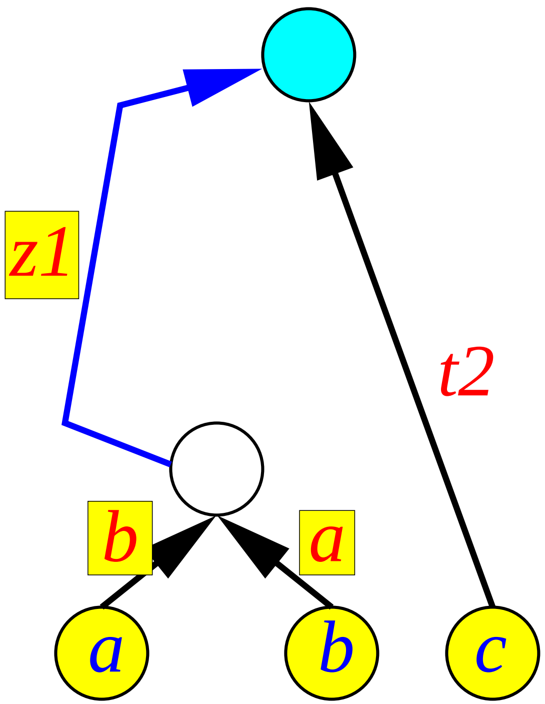
(local) Jacobians 16



edge eliminations: isolated vertex/edge

```
t1    = a*b
p1    = cos(t1)
t2    = sin(t1)
y     = t2*c
z1    = c * p1
```

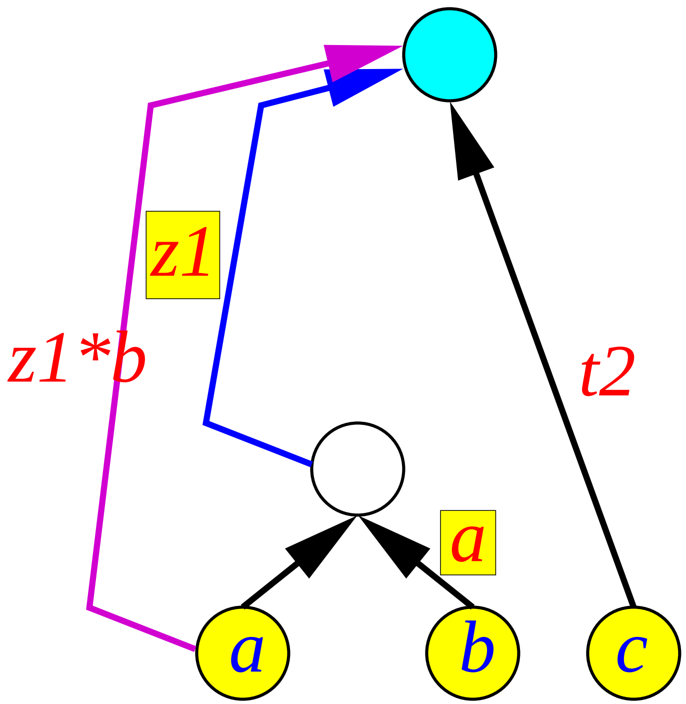
(local) Jacobians 17



edge eliminations: pick the next edge

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = c * p1
```

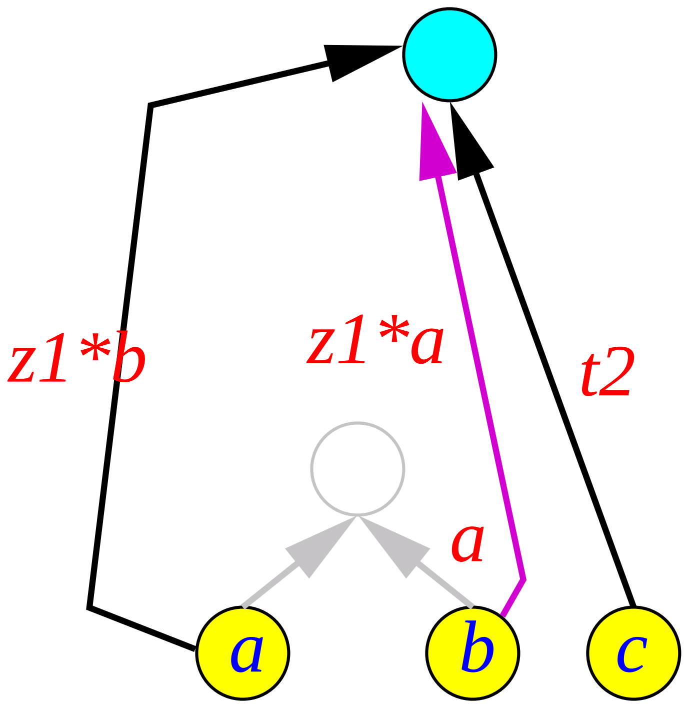
(local) Jacobians 18



edge eliminations: multiply edge labels for the first pair

```
t1    = a*b
p1    = cos(t1)
t2    = sin(t1)
y     = t2*c
z1    = c * p1
z2    = z1 * b
```

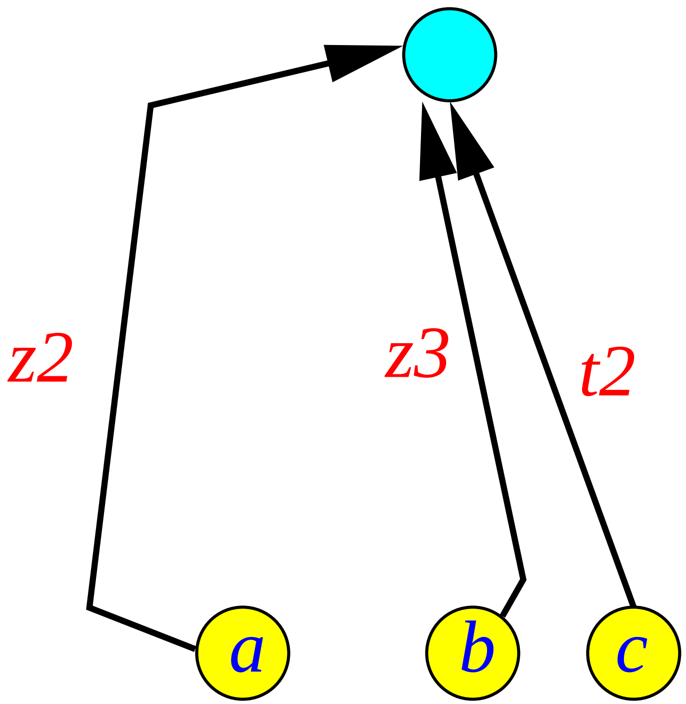
(local) Jacobians 19



edge eliminations: multiply edge labels for the second pair

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = c * p1
z2   = z1 * b
z3   = z1 * a
```

(local) Jacobians 20



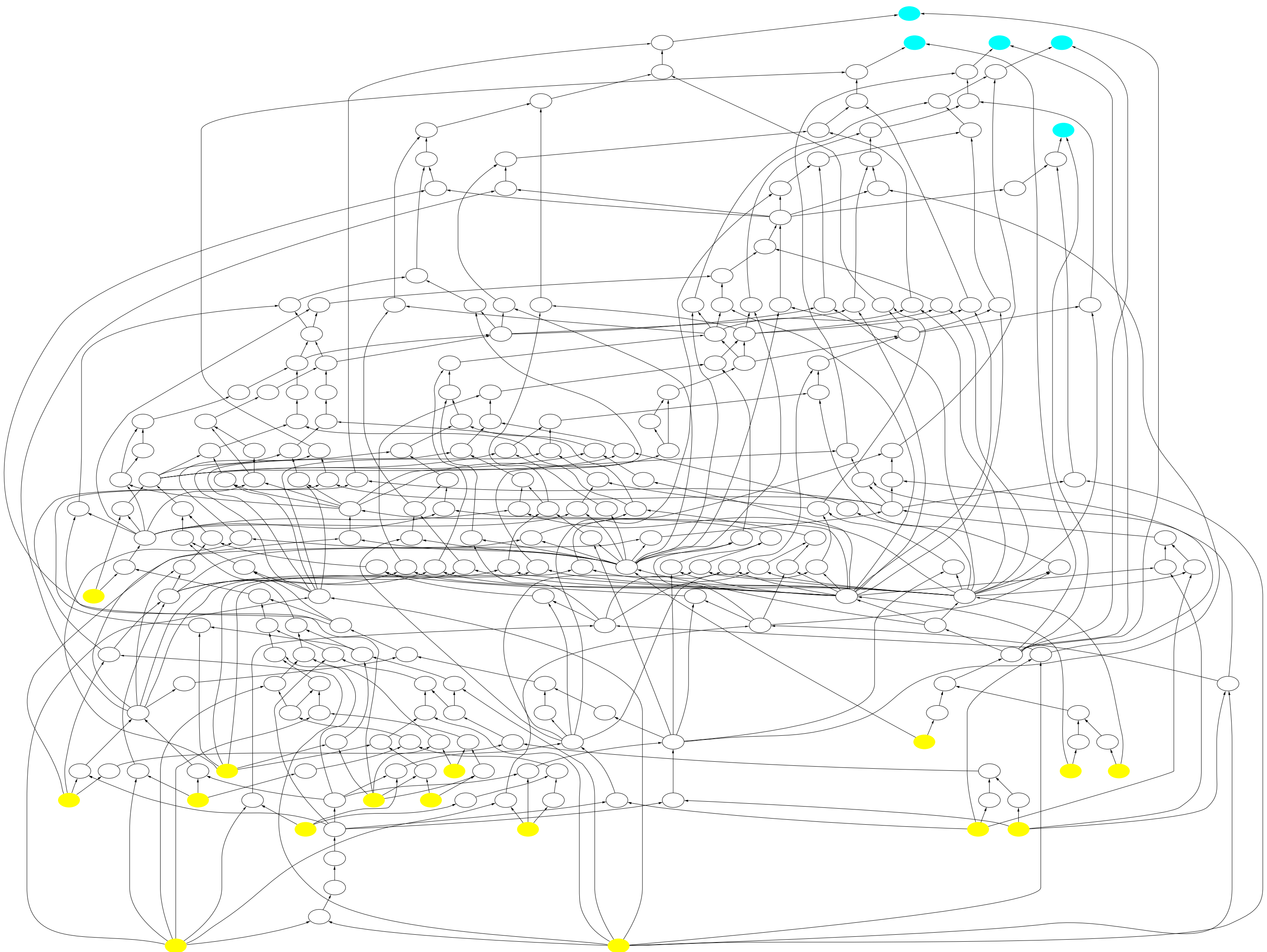
edge eliminations: bipartite graph,

done in 3 operations

```
t1   = a*b
p1   = cos(t1)
t2   = sin(t1)
y    = t2*c
z1   = c * p1
z2   = z1 * b
z3   = z1 * a
```

de**ad**

a bigger graph



heuristics

- pick an elimination target from an eligible set S
- each heuristic $h : S \mapsto S' \subseteq S$
- heuristic sequence $h_k(\ldots h_2(h_1(S))\ldots)$ with a tie-breaker h_k (such as “reverse”) returns a single elimination target
- eliminate target $\Rightarrow$ modified graph $\Rightarrow$ new S
- done when $S = \emptyset$
- operation count heuristics: Markowitz

heuristics

- pick an elimination target from an eligible set S
- each heuristic $h : S \mapsto S' \subseteq S$
- heuristic sequence $h_k(\ldots h_2(h_1(S))\ldots)$ with a tie-breaker h_k (such as "reverse") returns a single elimination target
- eliminate target $\Rightarrow$ modified graph $\Rightarrow$ new S
- done when $S = \emptyset$
- operation count heuristics: Markowitz

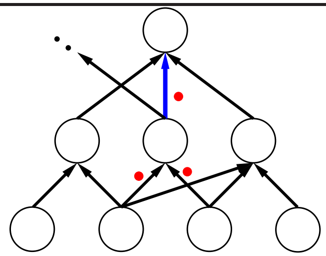
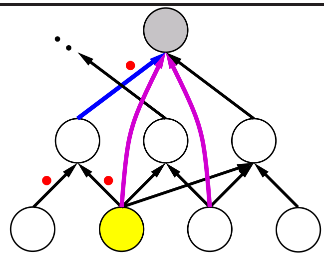
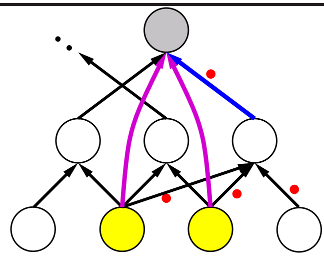
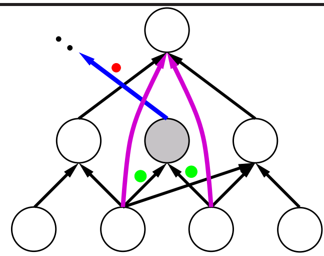
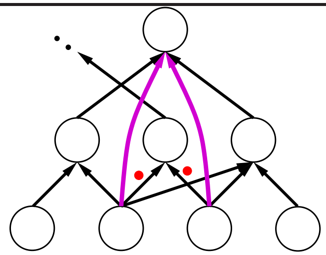
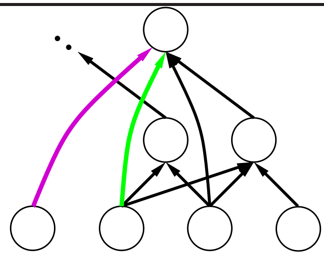
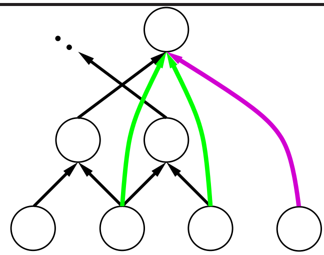
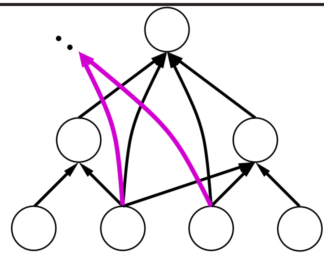
  Harry Max Markowitz, b. 1927, economics NP 1990,

  "Becoming an economist was not a childhood dream of mine."

- data locality: forward, reverse, sibling(s), pc, absorb
- forward (top sort): first mark all minimal vertices, mark vertices with all pred. marked (order based on undelying graph representation)
- reverse: reverse of forward

sibling heuristics

relate subsequent elimination target with respect to the variables occuring in the current elimination step:

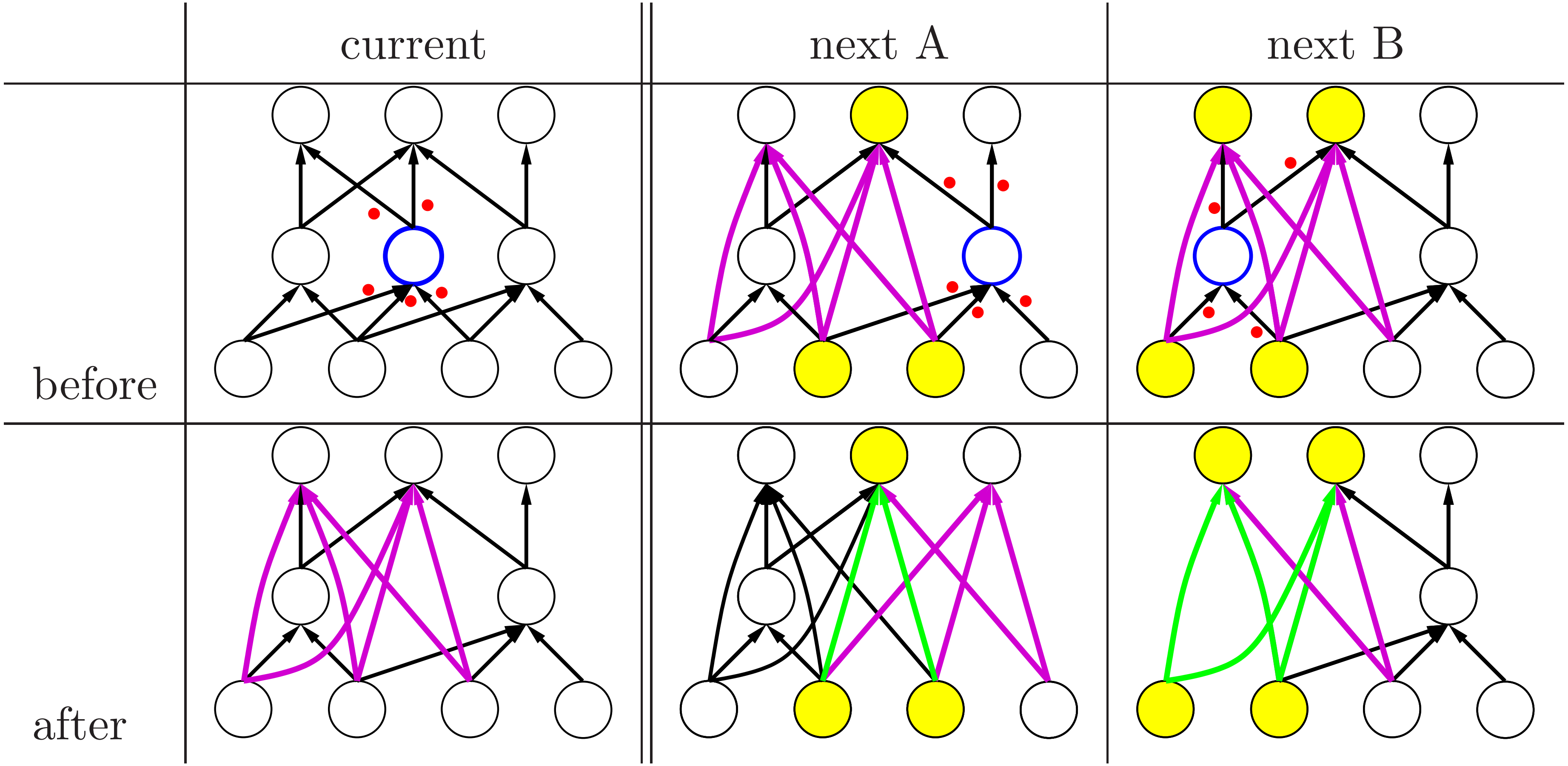
	current	next A	next B	vertex elimination
before				
after				

- same target, max number of source's predecessors, or
- same source, max number of target's successors

dreadful

sibling heuristics 2

for edge eliminations grouped into vertex eliminations (target is a vertex):



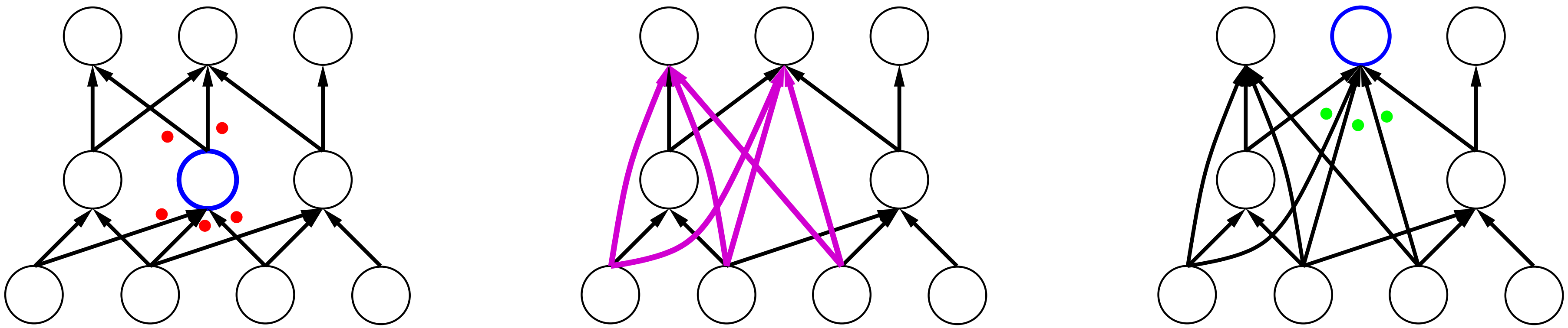
- max product of shared predecessors and successors

pc – vertex

- does not mean "nouvelle orthodoxie"

  (politically correct translation of political correctness purportedly given by the *Office Québécois de la Langue Française*)

- parent-child (or the other way round)

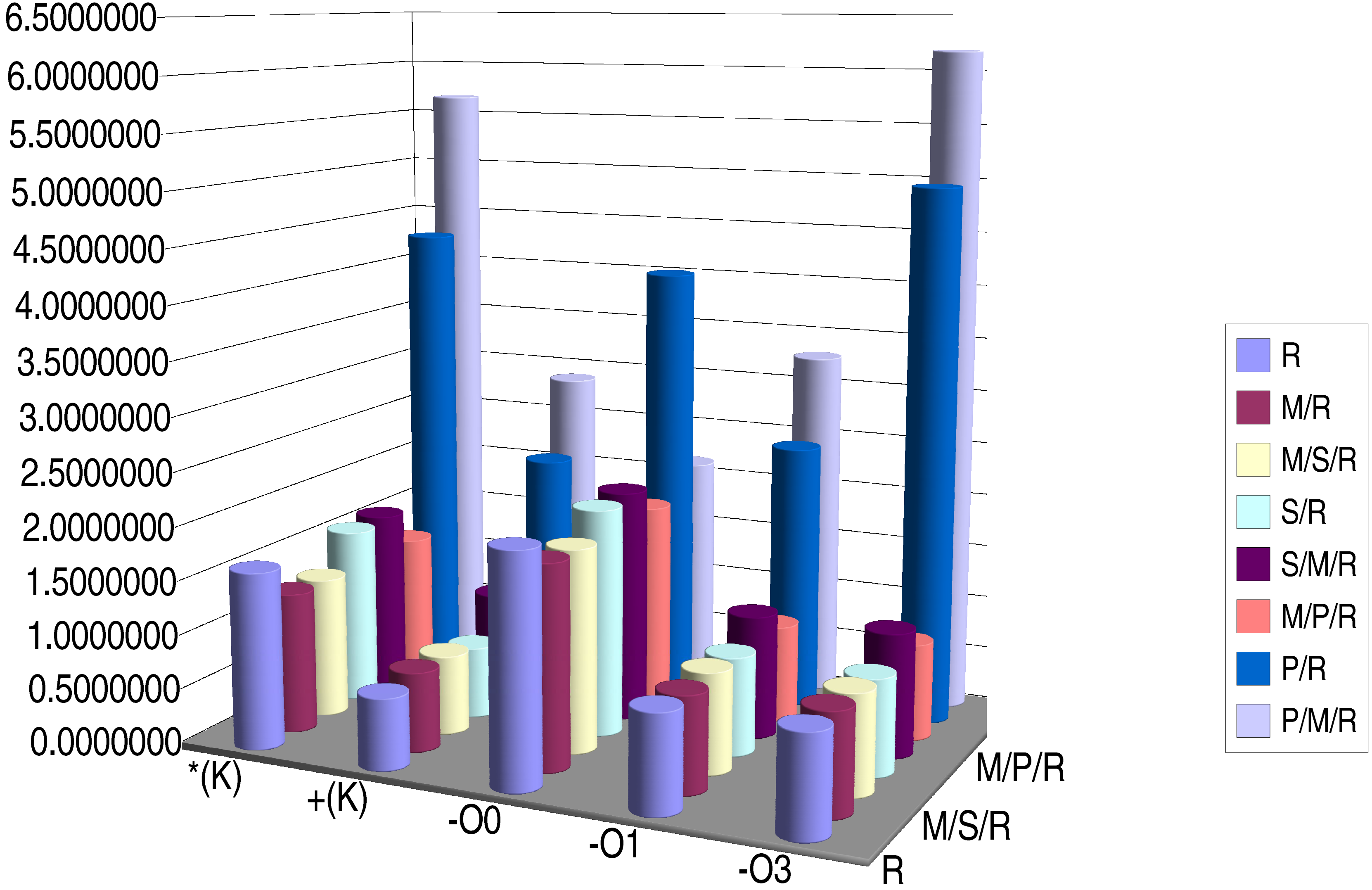


- but prefers targets with high Markowitz degree ⇒ sequence after Markowitz

malady

RF graph

RF results – vertex elimination

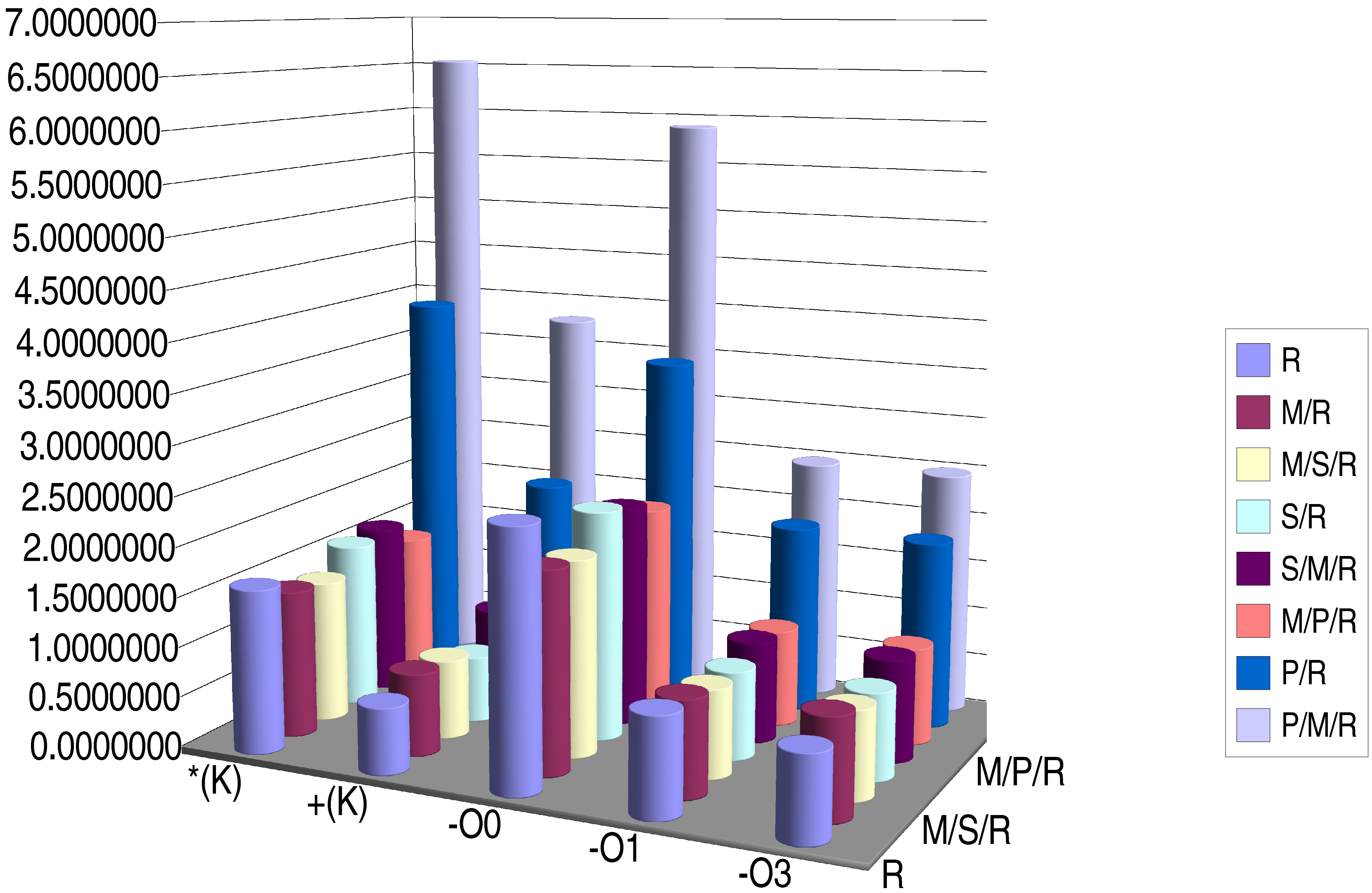


RF results – vertex elimination

heuristics	time*	mults	adds	comments
h_1: reverse	2.1346599 .91040001 .93199997	1639	664	reverse because 16 independents - 5 dependents
h_1: Markowitz h_2: reverse	1.8921801 .89185996 .94176003	1305	738	initially Markowitz degree 1, then 2 $\Rightarrow$ reverse until last 38 (5th last Markowitz degree 70)
h_1: Markowitz h_2: sibling h_3: reverse	1.8850400 .93387998 .93097997	1305	738	no siblings until the last 15%. (like popcorn)
h_1: sibling h_2: reverse	2.1185801 .91474003 .89746000	1639	667	23 siblings from 222 eliminations
h_1: sibling h_2: Markowitz h_3: reverse	2.1619000 1.1436000 1.1503800	1674	1032	
h_1: Markowitz h_2: pc h_3: reverse	1.9009200 .90116000 .89630003	1314	738	not much slower than the fastest
h_1: pc h_2: reverse	4.0542000 2.4809600 4.9855200	4298	2125	pc runs counter Markowitz
h_1: pc h_2: Markowitz h_3: reverse	2.1073880 3.2610002 6.1801598	5656	2855	pc runs counter Markowitz

* ifort on Linux/Intel flags: -O0 / -O1 / -O3

RF results – edge elimination

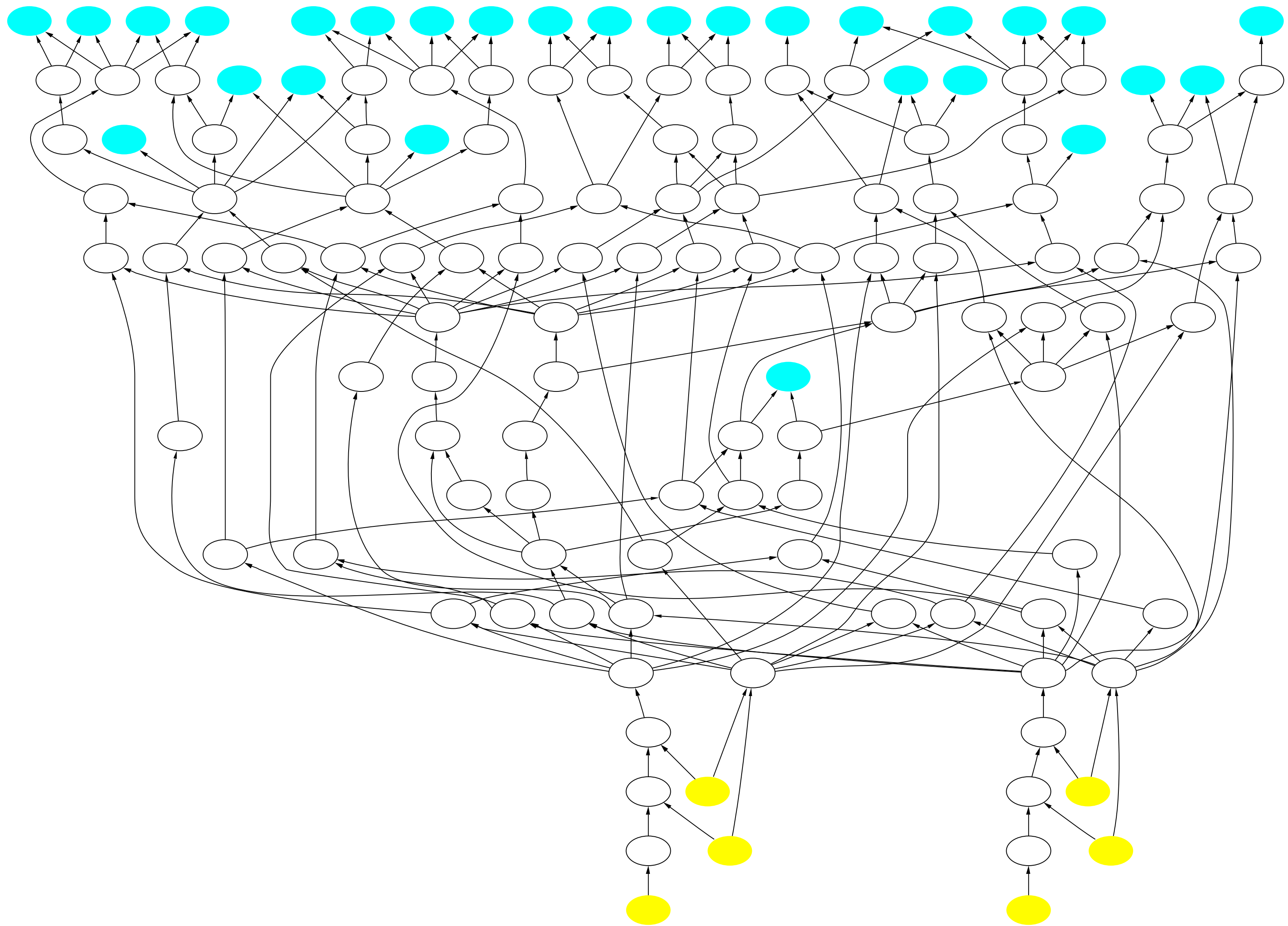


RF results – edge elimination

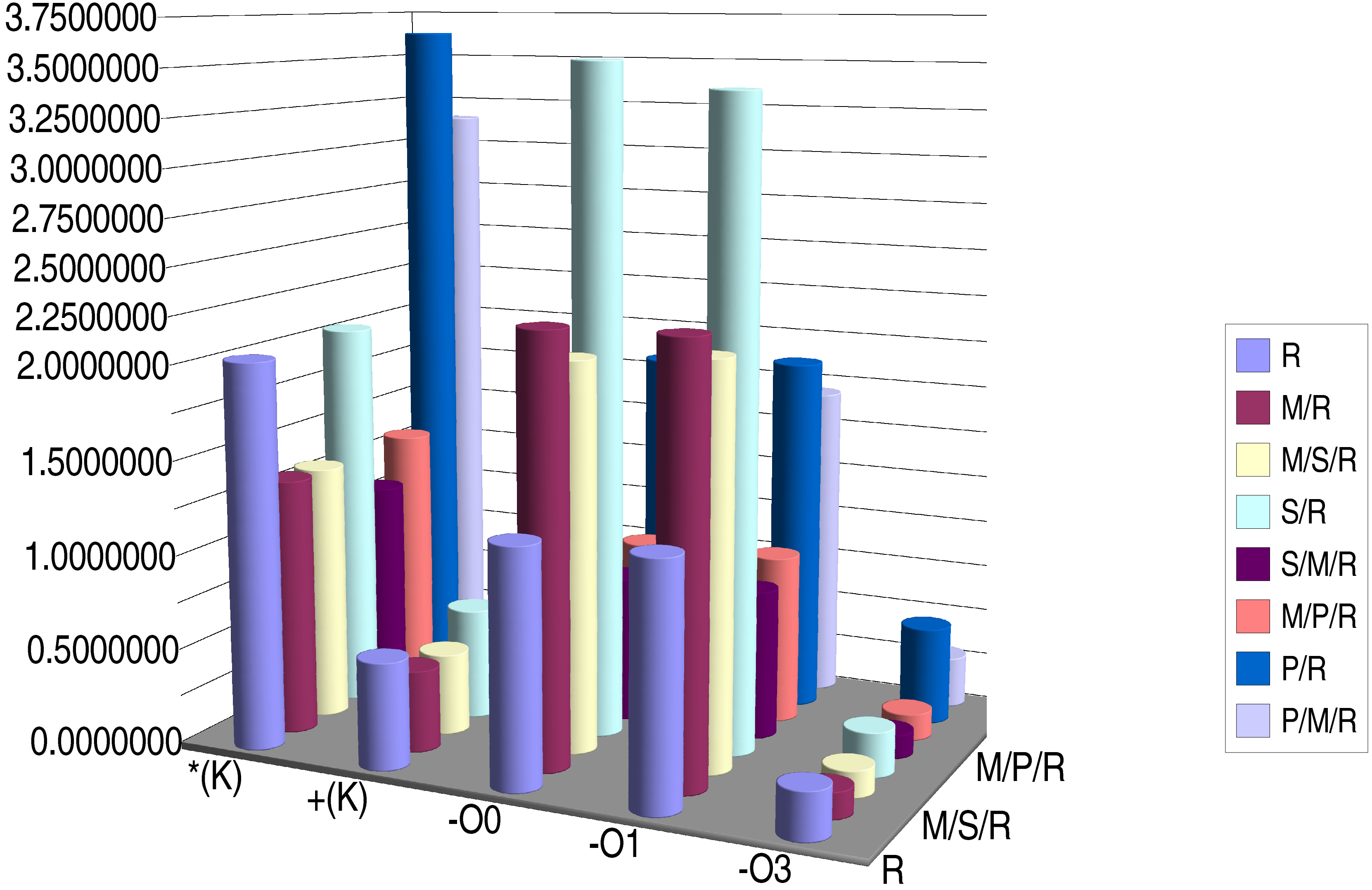
heuristics	time*	mults	adds	comments
h_1: reverse	2.5624000 .99068002 .85047996	1639	664	marginally better than vertex elimination
h_1: Markowitz h_2: reverse	2.0155800 .96147996 1.0048000	1472	824	
h_1: Markowitz h_2: sibling h_3: reverse	1.9675001 .88325996 .88478000	1420	795	
h_1: sibling h_2: reverse	2.3091199 .88675997 .87183998	1660	667	
h_1: sibling h_2: Markowitz h_3: reverse	2.2150200 1.0113000 1.0213200	1708	990	
h_1: Markowitz h_2: pc h_3: reverse	2.0704199 .97060001 .97131997	1469	824	
h_1: pc h_2: reverse	3.4875000 1.8960000 1.8893999	3931	2066	same behavior as in vertex elimination
h_1: pc h_2: Markowitz h_3: reverse	5.8903399 2.4457800 2.4552801	6532	3770	same behavior as in vertex elimination

* -O0 / -O1 / -O3

TM graph



TM results – vertex elimination

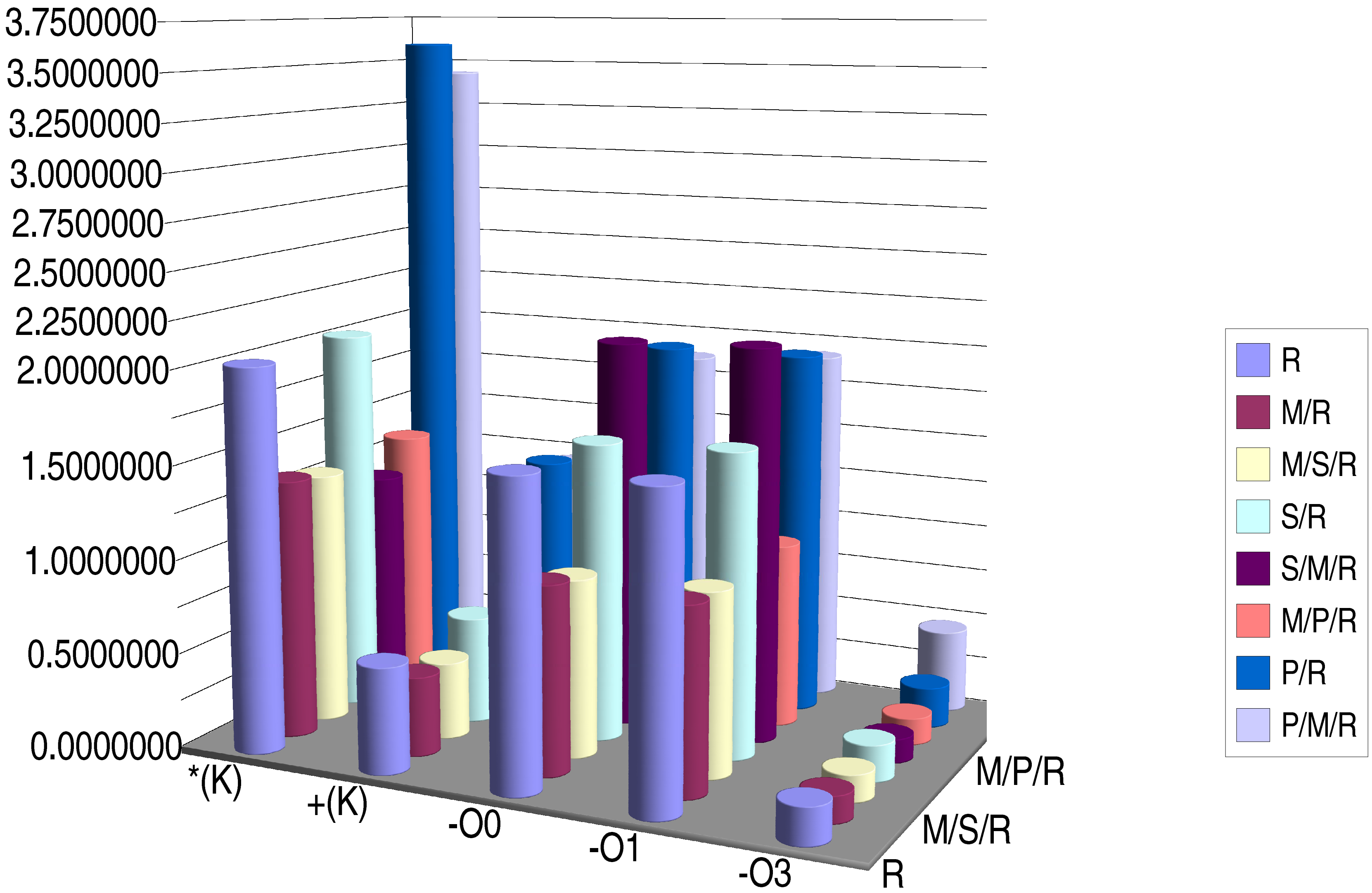


TM results – vertex elimination

heuristics	time*	mults	adds	comments
h_1: reverse	1.2493000 1.2736400 .25096001	2037	566	
h_1: Markowitz h_2: reverse	2.2586401 2.2754401 .13962000	1360	433	
h_1: Markowitz h_2: sibling h_3: reverse	2.0593399 2.1214601 .14244000	1360	433	
h_1: sibling h_2: reverse	3.5347799 3.4022600 .23806001	2065	590	
h_1: sibling h_2: Markowitz h_3: reverse	.77883997 .79069996 .12492000	1125	325	
h_1: Markowitz h_2: pc h_3: reverse	.89196001 .89843998 .14296000	1361	433	
h_1: pc h_2: reverse	1.8635399 1.8957400 .51779998	3633	1251	
h_1: pc h_2: Markowitz h_3: reverse	1.6529601 1.6788401 .26920000	3142	1069	

* ifort on Linux/Intel flags: -O0 / -O1 / -O3

TM results – edge elimination

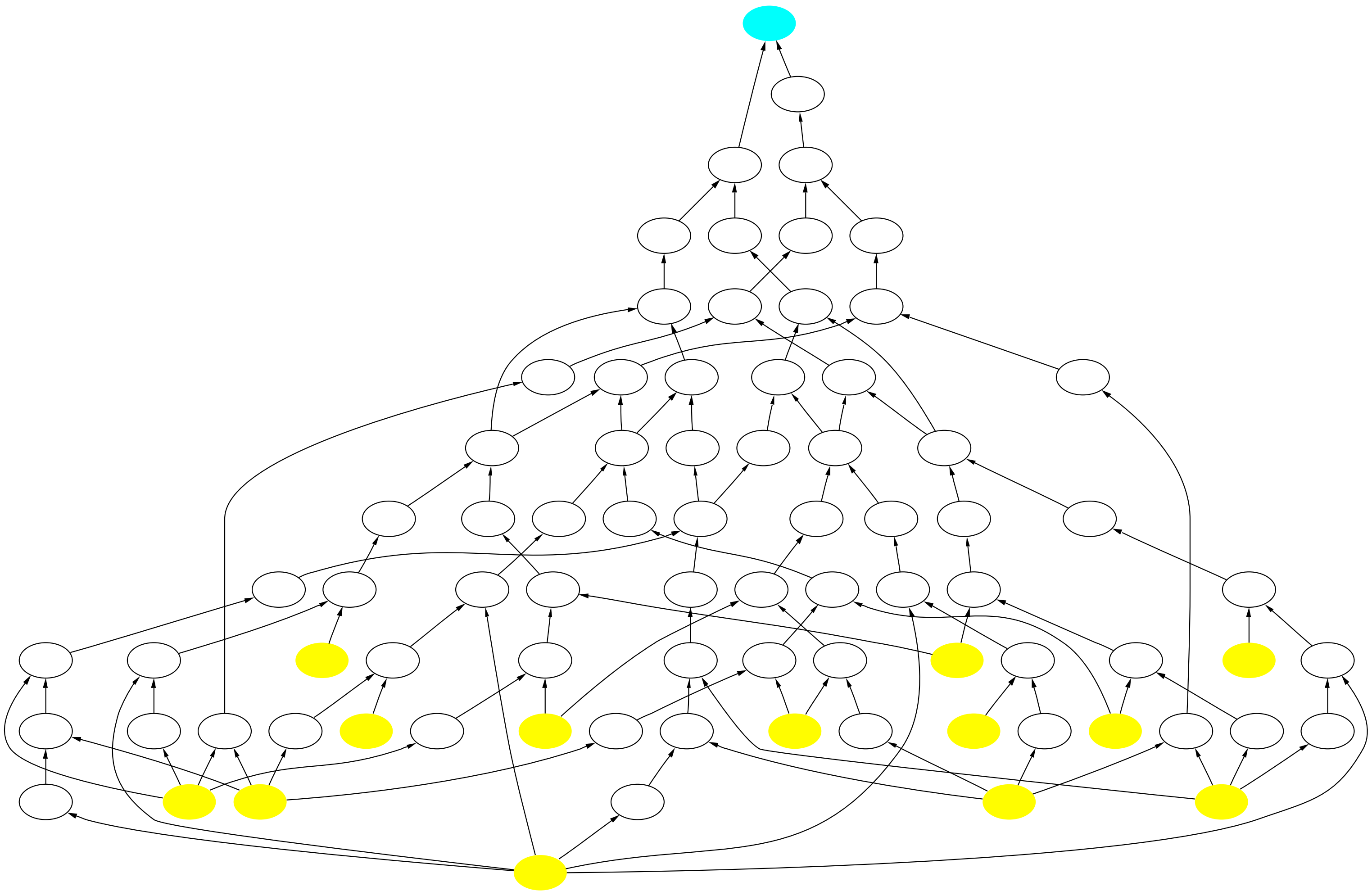


TM results – edge elimination

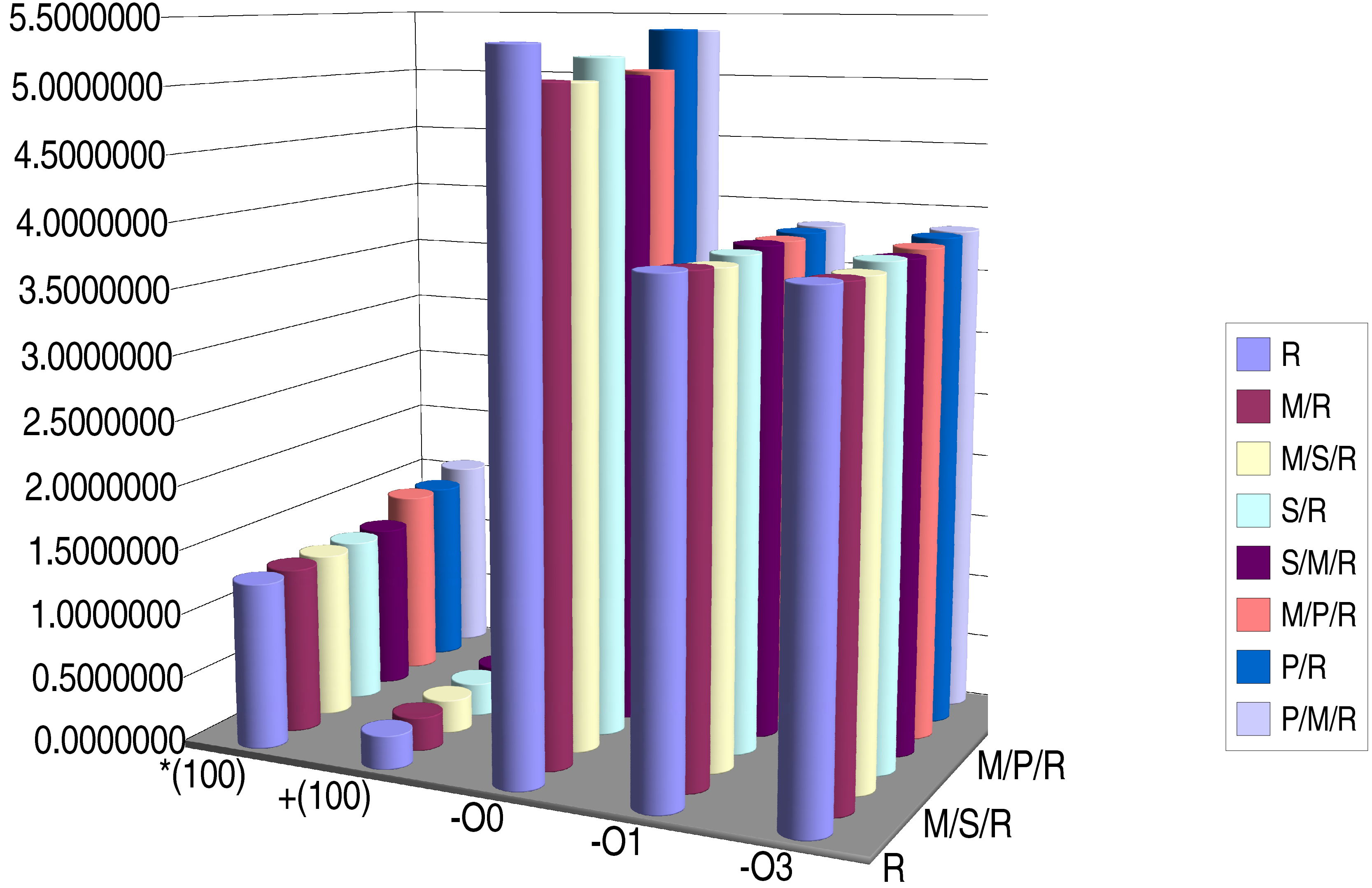
heuristics	time*	mults	adds	comments
h_1: reverse	1.6197801 1.6393400 .19731999	2037	566	
h_1: Markowitz h_2: reverse	1.0008000 .98913997 .14300000	1383	423	
h_1: Markowitz h_2: sibling h_3: reverse	.95034002 .97614002 .14446000	1347	411	
h_1: sibling h_2: reverse	1.5942800 1.6216600 .19540000	2055	572	
h_1: sibling h_2: Markowitz h_3: reverse	2.0748999 2.1038000 .12775999	1208	350	
h_1: Markowitz h_2: pc h_3: reverse	.99008003 .98853998 .14446000	1383	423	
h_1: pc h_2: reverse	1.9577399 1.9643800 .21748001	3599	1243	
h_1: pc h_2: Markowitz h_3: reverse	1.8553401 1.9116200 .44260000	3431	1185	

* -O0 / -O1 / -O3

DC graph



DC results – vertex elimination

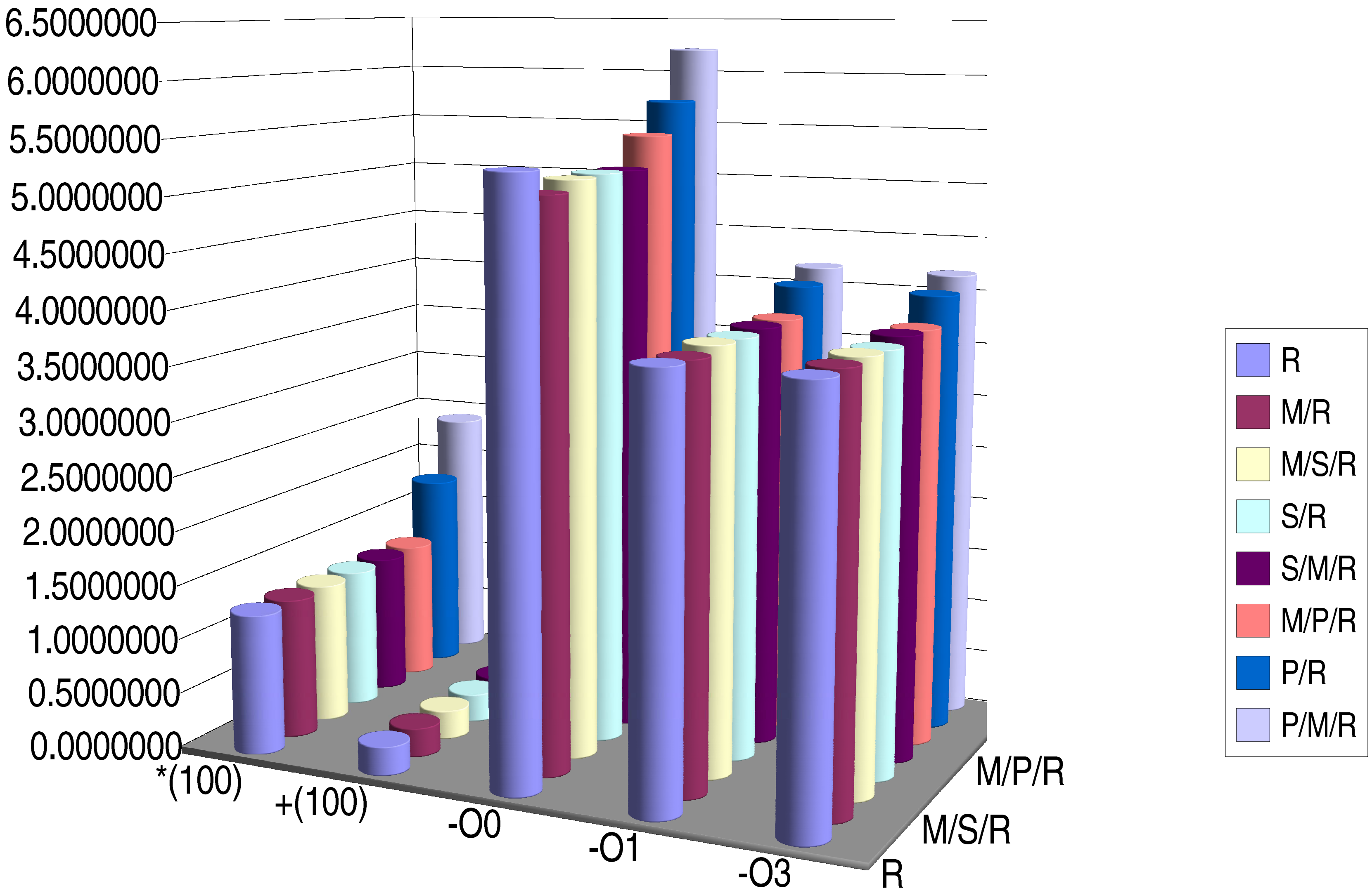


DC results – vertex elimination

heuristics	time*	mults	adds	comments
h_1: reverse	5.3393600 3.8298000 3.8138602	129	26	
h_1: Markowitz h_2: reverse	5.0617800 3.7949601 3.7798200	129	26	
h_1: Markowitz h_2: sibling h_3: reverse	5.0407401 3.7596801 3.7627800	129	26	
h_1: sibling h_2: reverse	5.2046598 3.7990601 3.8121401	129	26	
h_1: sibling h_2: Markowitz h_3: reverse	5.0470800 3.8194401 3.7762398	129	26	
h_1: Markowitz h_2: pc h_3: reverse	5.0745999 3.8050199 3.8028200	147	26	
h_1: pc h_2: reverse	5.3799398 3.8230599 3.8355800	145	28	
h_1: pc h_2: Markowitz h_3: reverse	5.3634802 3.8308601 3.8415601	154	31	

* ifort on Linux/Intel flags: -O0 / -O1 / -O3

DC results – edge elimination



DC results – edge elimination

heuristics	time*	mults	adds	comments
h_1: reverse	5.3263200 3.8164599 3.8144000	129	26	
h_1: Markowitz h_2: reverse	5.0944600 3.7922000 3.8168600	129	26	
h_1: Markowitz h_2: sibling h_3: reverse	5.1780000 3.8327998 3.8279799	129	26	
h_1: sibling h_2: reverse	5.1891999 3.8130998 3.7996801	129	26	
h_1: sibling h_2: Markowitz h_3: reverse	5.1781401 3.8280599 3.8433001	129	26	
h_1: Markowitz h_2: pc h_3: reverse	5.4553599 3.8289199 3.8206799	129	26	
h_1: pc h_2: reverse	5.7405199 4.0686002 4.0551400	184	41	
h_1: pc h_2: Markowitz h_3: reverse	6.2111401 4.1804400 4.1750201	238	49	

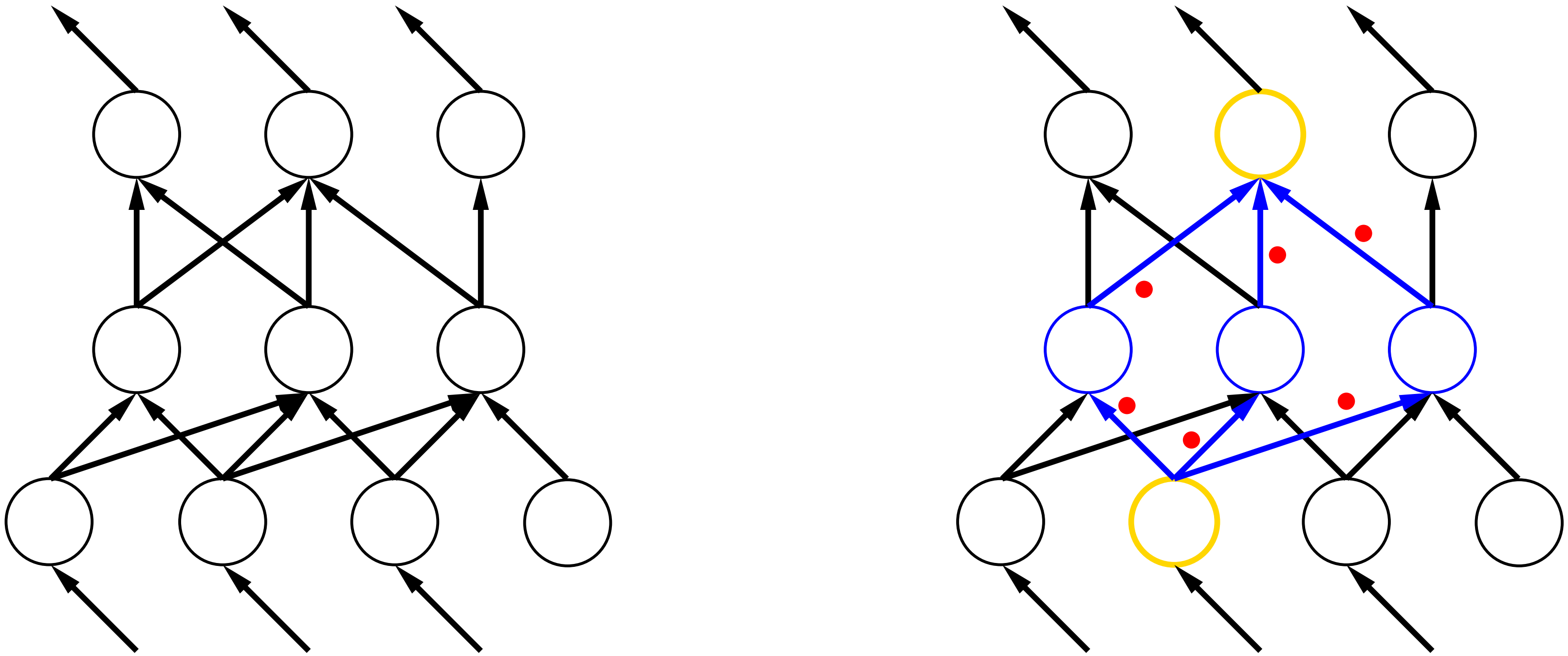
* -O0 / -O1 / -O3

absorb 1

- pick elimination targets such that absorption happens
- J. Pryce (Nov/04): regroup operations

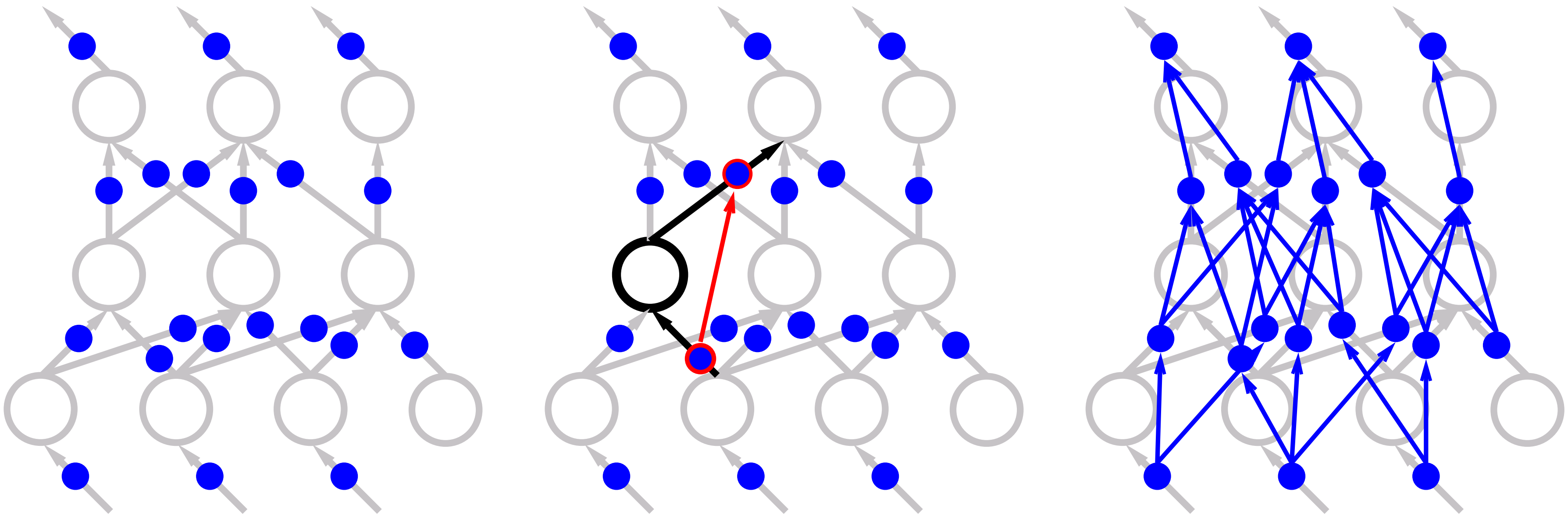
  $a = a + bc$; $e = e + fg$; $h = h + ij$; $a = a + kl$; $m = m + no$; $a = a + pq$

  based on the absorbing a to $a = a + bc + kl + pq$
- not representable in the computational graph



absorb 2

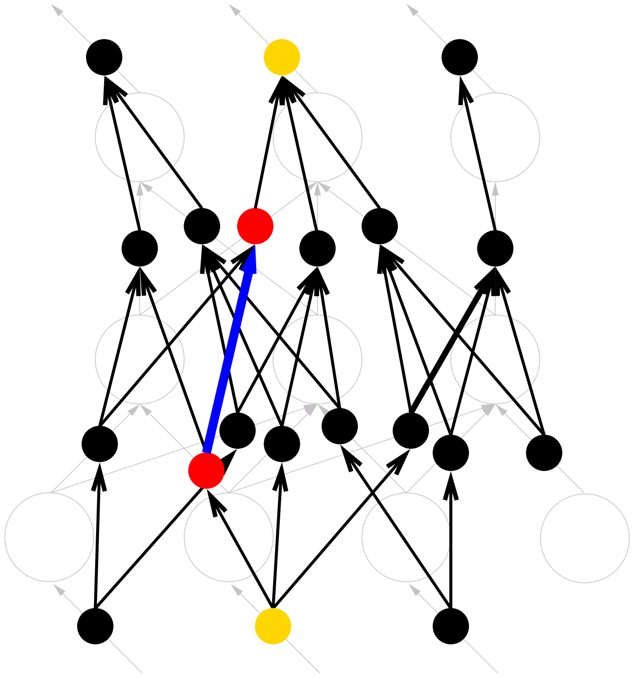
- pick elimination targets such that absorption happens
- J. Pryce (Nov/04): regroup operations
  $a = a + bc;\ e = e + fg;\ h = h + ij;\ a = a + kl;\ m = m + no;\ a = a + pq$
  based on the absorbing a to $a = a + bc + kl + pq$
- not representable in the computational graph $\Rightarrow$ directed line graph

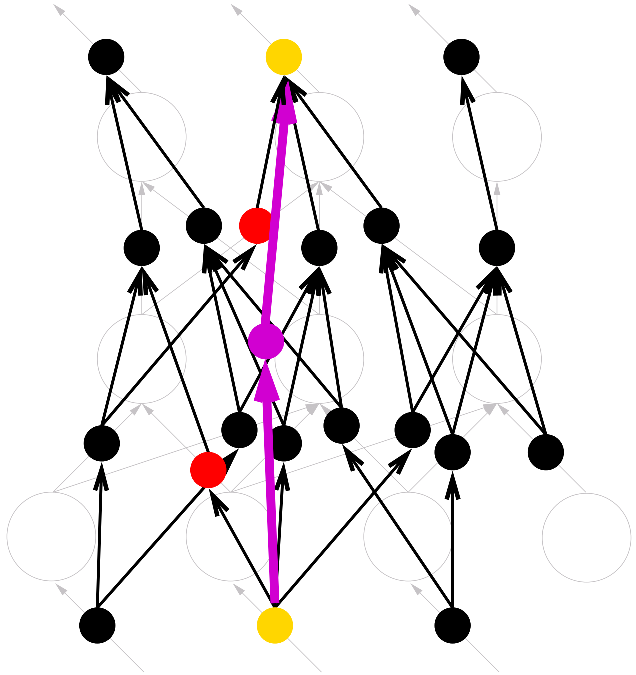


mad

absorb 3

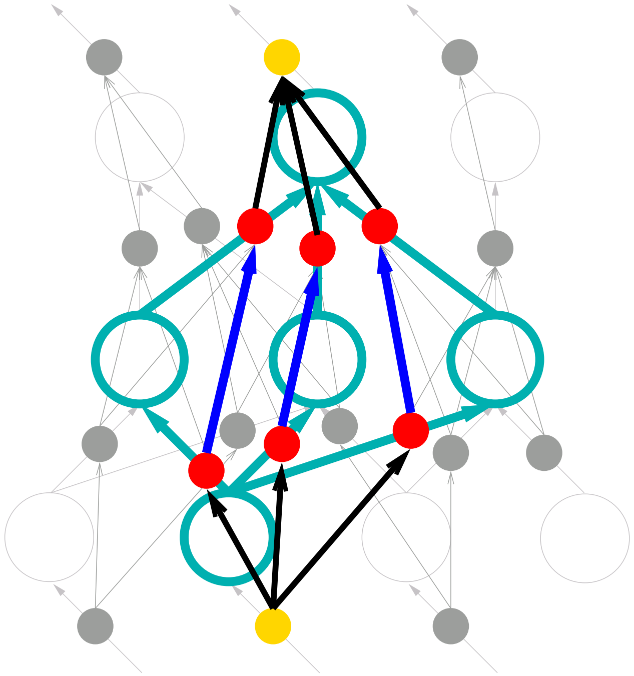
- pick elimination targets such that absorption happens
- J. Pryce (Nov/04): regroup operations
  $a = a + bc$; $e = e + fg$; $h = h + ij$; $a = a + kl$; $m = m + no$; $a = a + pq$
  based on the absorbing a to $a = a + bc + kl + pq$
- not representable in the computational graph $\Rightarrow$ directed line graph

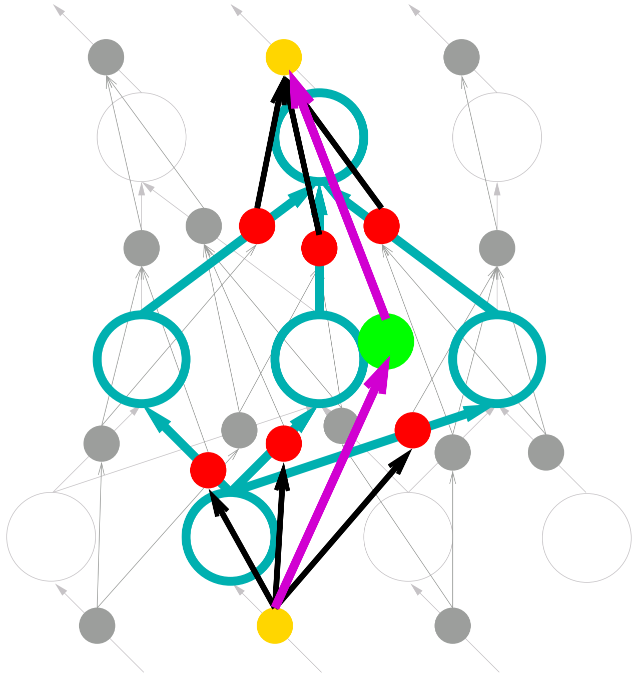




absorb 4

- pick elimination targets such that absorption happens
- J. Pryce (Nov/04): regroup operations
  $a = a + bc$; $e = e + fg$; $h = h + ij$; $a = a + kl$; $m = m + no$; $a = a + pq$
  based on the absorbing a to $a = a + bc + kl + pq$
- not representable in the computational graph $\Rightarrow$ directed line graph





implementation & conclusions

- ACTS project Argonne National Laboratory, MIT, Rice University, RWTH Aachen
- numerical models (design optimization, chemical engineering, oceanography)
- transformations: automatic differentiation, interval, ensemble computations (uncertainty estimates)
- Fortran (C/C++, Matlab, Java)
- website: `www.mcs.anl.gov/openad`

- in adjoint code context effects are smaller than checkpointing / taping improvements
- data locality heuristics doesn't improve things (compiler gets that part right)
- op count does improve things (compiler can't improve)
- late stage improvements, but automated
- consistent through compiler optimization
- before final conclusion: more examples, more compilers, constant folding
- future potential: vector operations

adieu!